Name: Anushka M Mulay

Class: TE IT (A)

Roll No: 35040

Batch: B

Group: C

**Assignment No. 5**

**Title:** Aggregation and indexing.

**Problem Statement**: Implement the aggregation and indexing with suitable example in MongoDB. Demonstrate the following:

* Aggregation framework
* Create and drop different types of indexes and explain () to show the advantage of the indexes.

**Requirements:** MongoDB.

**Prerequisites:** Basic knowledge about mongoDB.

**Theory:**

1. **Aggregation framework**

The aggregation framework lets you transform and combine documents in a collection.

Basically, you build a pipeline that processes a stream of documents through several

building blocks: filtering, projecting, grouping, sorting, limiting, and skipping.

For example, if you had a collection of magazine articles, you might want find out who

your most prolific authors were. Assuming that each article is stored as a document in

MongoDB, you could create a pipeline with several steps:

* Project the authors out of each article document.
* Group the authors by name, counting the number of occurrences.
* Sort the authors by the occurrence count, descending.
* Limit results to the first five.

Each of these steps maps to a aggregation framework operator:

**1**. {"$project" : {"author" : 1}}

This projects the author field in each document.

The syntax is similar to the field selector used in querying: you can select fields to

project by specifying "fieldname" : 1 or exclude fields with "fieldname" : 0.

After this operation, each document in the results looks like: {"\_id" : id, "au

thor" : "authorName"}. These resulting documents only exists in memory and

are not written to disk anywhere.

**2**. {"$group" : {"\_id" : "$author", "count" : {"$sum" : 1}}}

This groups the authors by name and increments "count" for each document an

author appears in.

First, we specify the field we want to group by, which is "author". This is indicated

by the "\_id" : "$author" field. You can picture this as: after the group there will

be one result document per author, so "author" becomes the unique identifier

("\_id").

The second field means to add 1 to a "count" field for each document in the group.

Note that the incoming documents do not have a "count" field; this is a new field

created by the "$group".

At the end of this step, each document in the results looks like: {"\_id" : "author

Name", "count" : articleCount}.

**3**. {"$sort" : {"count" : -1}}

This reorders the result documents by the "count" field from greatest to least.

**4.** {"$limit" : 5}

This limits the result set to the first five result documents.

1. **Indexing and types of indexing**

A database index is similar to a book’s index. Instead of looking through the whole book,

the database takes a shortcut and just looks at an ordered list that points to the content,

which allows it to query orders of magnitude faster.

A query that does not use an index is called a table scan (a term inherited from relational

databases), which means that the server has to “look through the whole book” to find

a query’s results. This process is basically what you’d do if you were looking for infor‐

mation in a book without an index: you start at page 1 and read through the whole thing.

In general, you want to avoid making the server do table scans because it is very slow

for large collections.

**Types of indexing**

* Unique Indexes
* Compound unique indexes
* Sparse Indexes
* Full-Text Indexes

1. **Explain() function**

explain() gives you lots of information about your queries. It is one of the most important diagnostic tools there is for slow queries. You can find out which indexes are being used and how by looking at a query’s explain. For any query, you can add a call to explain() at the end (the way you would add a sort() or limit(), but explain() must be the last call).

There are two types of explain() output that you’ll see most commonly: indexed and

non-indexed queries. Special index types may create slightly different query plans, but most fields should be similar. Also, sharding returns a conglomerate of explain()s

as it runs the query on multiple servers.

The most basic type of explain() is on a query that doesn’t use an index.

**Conclusion:** Thus, in this assignment we studied about Aggregation and indexing.